ORTA DOĞU TEKNİK ÜNİVERSİTESİ

KUZEY KIBRIS KAMPUSU

CNG 352 | ER Diagram

Briddgy

Our Team

- Rustam Quliyev 2250470
- Orkhan Salahov 2193191

Table of Contents

What is Briddgy ?

Our project for this semester is Briddgy – Social Network for Peer-to-peer deliveries. Using Briddgy, travelers can add their trip details and earn money by delivering, and those who order will get the opportunity to send or receive their items quickly and cheaper. Users can login/signup to our platform, publish their trips and orders. All the users can see all the trips and orders in the platform. Moreover, the users can search through all the matching orders for their trips and vice versa. Our platform will also support chat functionality. A user can start conversation with any other user and send proposals to work together. Proposals are basically contracts for both parties(traveler and ordered) to work together. Anyone with matching order/trip and make proposal to someone else's trip/order. Chat component will play the central role in proposal management, since only through chat will the users be able to propose and accept proposals. There will also be reviews, ratings and statistics of the users. In order to maintain maximum possible security, users must verify their emails and phone numbers. We will also have autocomplete feature while selecting the origin and destination cities of trip and order. Also users must be notified if someone made a proposal to them.

Data Requirements

Before we start explaining the data requirements, couple of words that we are going to use. A traveler is a person that has a trip in system. An orderer is a person who has an order in the system.

Users

To become a user of Briddgy, a person must register. The data stored on users includes name, surname, email, password. Also, we store the join date, last_online date, Boolean which shows whether a user is_online, is_staff, is_active, is_email_verified, is_number_verified, is_photo_verified. Each user will have a ranking field which gets a default 0 when user registers. In order to use the system the user can login by entering his/her email and password. We are going to have token for each login operation, and clients are going to send this token for authorization purposes. Each user can have a profile picture which can be changed in the profile page. A user can be a traveler(have a trip) or deliverer(have an order) or both. The system will also have statistics of the users, such as number of trips, number of orders, total earning and etc.

Trips

A trip is added by a user by specifying date (departure), weight_limit, source_city, destination_city. Each trip can have contracts with orders. Having a contract with order implies that the orders will be delivered with this particular trip. Contract is first offered by one party (traveler or orderer) through the Chat Interface(there will be a dedicated button to propose contract). Let's call him initiator party. System will automatically generate a message that will be delivered to the other party and ask whether to accept or discard the contract request. Upon accepting the two way contract will be stored in system with the signature_date.

Orders

An order is added by a user, by specifying a title, weight, reward (price user willing to pay for delivery), description, source_city, destination_city. Note that Orders are independent of Trip, in other words orders can be added to the system without a trip. Orders also has a relation with Contracts, an order can only have one contract. Orders have a requested_date field which takes today's date by default. Moreover, the creator can upload any number of images of its order to promote it better.

Chat Rooms

There are many rooms in the system, which have uuid, creation_time, last_modification_time, and many members. Any user can start a new chat room with any other user in the system. Although the system is going to have 2 person rooms, it can be easily scaled by adding any number of members.

Messages

There are many messages in a particular room which is sent by a particular user. Each message will have a text, and a list of recipients. If the user reads messages of a chat room, for all those messages the user is going to added to recipients list. It will be used to determine whether the message is read or not.

Cities

Although implicitly mentioned in Trips and Orders, the system will have a dictionary of allowed cities. Each city is going have name, country, latitude, longitude and etc. The fields depend on the choice of open dataset to be chosen. The system will suggest Autocorrection based on this dictionary once the user starts entering the city name.

Ranking

We have mentioned that each user is going to have a ranking, but how it is calculated ? Each user which has participated in a deal(contract) with other user, can rank him once. The rank will be in the range of [1..5] and will have the author and ranked user. System won't allow second ranking for the same user pairs, however, a user can overwrite his previous rank. Upon ranking the average sum of all the ranks for ranked user will be calculated ,and his/her rank will be updated accordingly.

Review

Each user which has participated in a deal(contract) with the other user, can give him reviews. There are no restrictions in number of reviews to be given. Each review will have a plain text, an author and reviewed user. Users can view each other's reviews by entering to their profiles.

Note that review and rank are between 2 users not between user and contract.

Pictures & ContentTypes

There are pictures of users, tasks and possibly messages. So instead of creating 3 spearate relations we chose to implement some sort of Generic Relations that is supported by most modern frameworks. Idea is that we will have a table **ContentTypes** which is going to have all other table names as relations. For example: [(1,"User"),(2,"Order")]. And Pictures will belong to some content type with specific ID. I will discuss it further during the translation part.

Transaction Requirements

1. *Login the User. (Make sure the password is right, give the user token if exists or create new one)=>SELECT,JOIN,CREATE*
2. *Get my Users' details. (Make sure token is correct, get the user details with his/her avatar_photo)=>SELECT, JOINx2*
3. *Get all the chatrooms. (Make sure token is correct, get all the chatroom for the user)=>SELECT, JOINx2*
4. *Get all the messages for the chatroom. (Make sure token is correct, make sure user is in the given room, return the messages)=> SELECT, JOINx2*
5. *Get all the orders according to given filters. (Make sure token is correct, match the orders to filter restrictions, return orders with their photos)=>SELECT,JOIN*
6. *Get all the trips according to given filters. (Make sure token is correct, match the trips to filter restrictions)=>SELECT*
7. *Get my users trips. (Make sure token is correct, find my users trips and return)=> SELECT, JOIN*
8. *Get my users orders. (Make sure token is correct, find my users orders their pictures, return)=>CREATE, JOINx2*
9. *Create new Order. (Make sure token is correct, Validate all the fields, create images and store in images table, update statistics for the user)=> SELECT, JOINx3*
10. *Create new Trip. (Make sure token is correct, Validate all the fields, update statistics for the user) => CREATE, JOINX2*
11. *Write a Message. (Make sure token is correct, add messages to the room if it exists or create new chatroom, push messages to the rooms socket)=>CREATEx2*
12. Make a new Contract a.k.a. Proposal. (Make sure token is correct, make sure chatroom exists, create contract object, create notification for other user)=> CREATEx2
13. Accept proposed Contract. (Make sure token is correct, make sure chatroom exists, make sure contract exists, udate contract object, create notification, update message)=>UPDATEx2,CREATE
14. Get all the orders suitable for the trip.(Make sure token is correct, Find the trip, Find all the matching orders with their images, return)=> SELECTx2, JOINx3
15. Get matching cities to the => SELECT, LIKE

ER Diagram

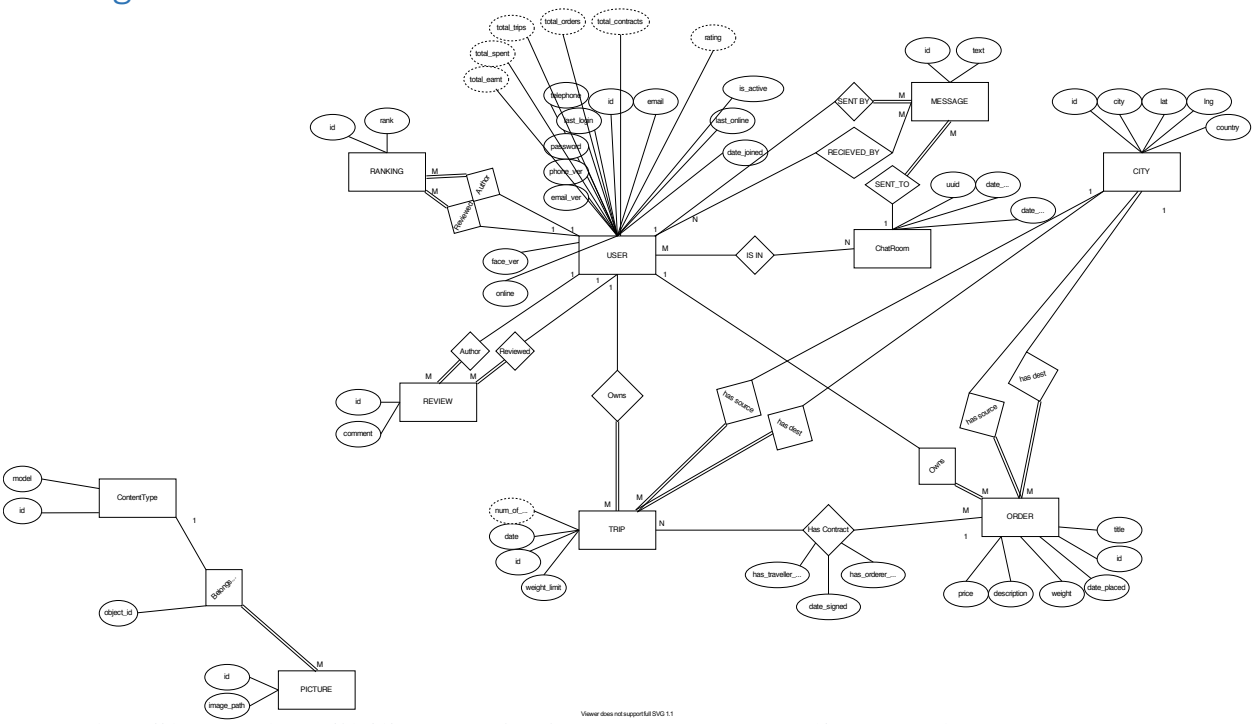

Figure 1 https://drive.google.com/file/d/1amGp0jQfWtgfgOxOiPn_VAVKG4QD-4Cl/view?usp=sharing

Alternatives & Assumptions

- We are going to assume that each trip will have single source and destination.
- We decided to not have any weak entities, since as most social networks do we would like to have the users' data even after they delete their profile for security reasons. For example if a user will make a contract with another user, then delete his profile we still need to have access to their chat and know the fact that those 2 have had a contract made.
- We are going to store any image in the file system or remote file system, and only store the path in the database.

Relations

Part A

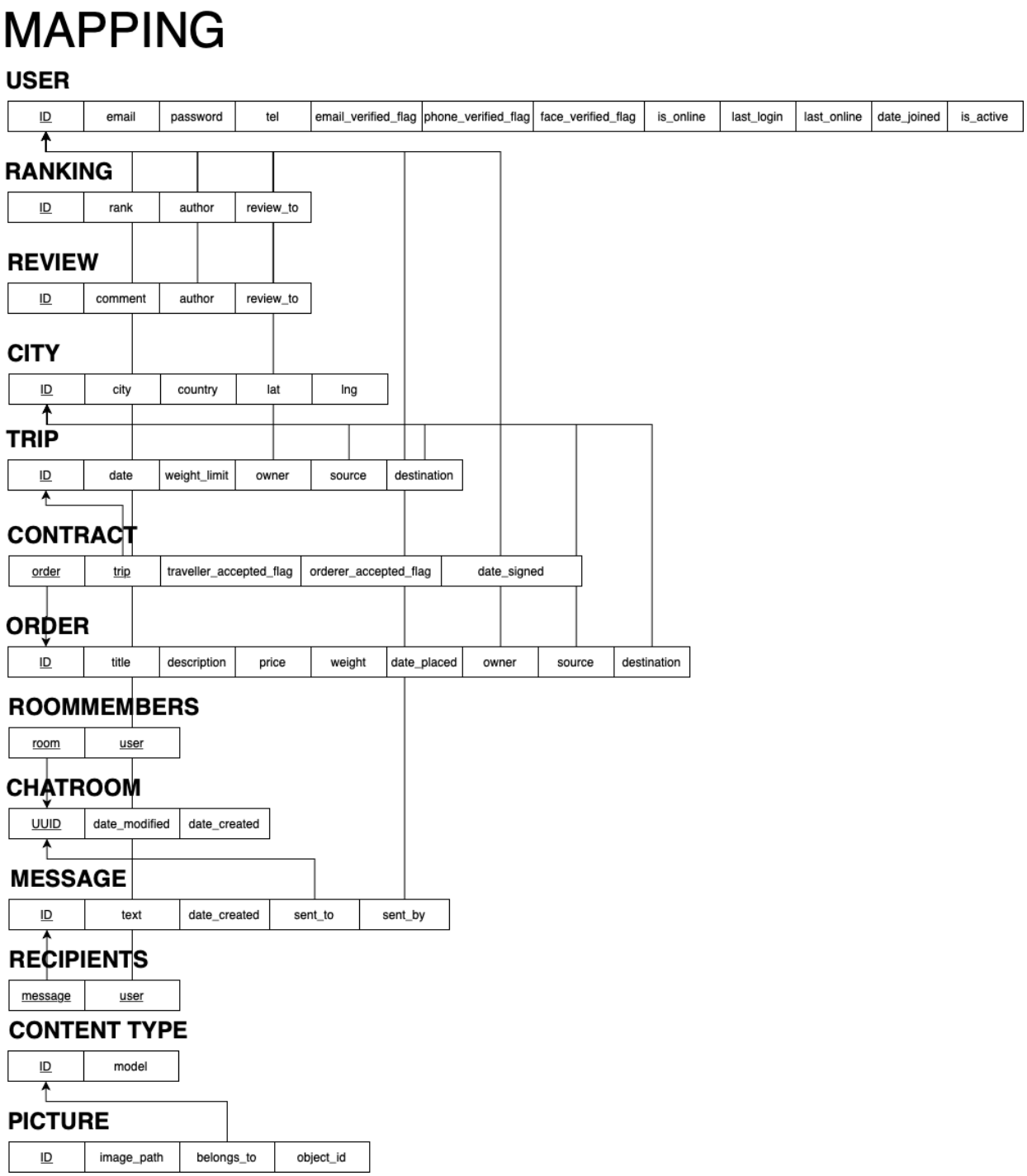

Figure 2 https://drive.google.com/file/d/1mPJgPrvAwxgehIpCxDAgf3HiCO-ULtdj/view?usp=sharing

PART B

Functional Dependency

USER In BCNF Since ID,Email are 2 superkeys of the relation

ID	email	password	tel	email_verified_flag	phone_verified_flag	face_verified_flag	is_online	last_login	last_online	date_joined	is_active

RANKING In BCNF Since {id,[author,review_to]} are superkeys

ID	rank	author	review_to

REVIEW In BCNF

ID	comment	author	review_to

CITY In BCNF Since (city,country) are superkeys

ID	city	country	lat	lng

TRIP In BCNF

ID	date	weight_limit	owner	source	destination

CONTRACT In BCNF

order	trip	traveller_accepted_flag	orderer_accepted_flag	date_signed

ORDER In BCNF

ID	title	description	price	weight	date_placed	owner	source	destination

ROOMMEMBERS In BCNF

room	user

CHATROOM In BCNF

UUID	date_modified	date_created

MESSAGE In BCNF since (triplet) is a superkey

ID	text	date_created	sent_to	sent_by

RECIPIENTS In BCNF

message	user

CONTENT TYPE In BCNF

ID	model

PICTURE In BCNF

ID	image_path	belongs_to	object_id

Figure 3https://drive.google.com/file/d/1mPJgPrvAwxgehIpCxDAgf3HiCO-ULtdj/view?usp=sharing

PART C

All relations are in Boyce-Codd Normal Form.

PART D

There are no Multivalued Nontrivial Dependencies

PART E

No. All the Entities are in their correct state.

PART F

If we look at the common case usage of the database resources. Intuitively we can see that users will be browsing through others Tasks, Orders and Profiles most of the time, and seldom will they create tasks and orders*. In such case where the amount of updates in the database is relatively low we believe that some kind of redundancy should exist in the database. For that reason we have done the following.

- Took all the Users statistics in a separate table, so after each user update the applications is going to update them. Therefore no resources will be spent on aggregating them.
- Make user rating as an attribute of the user. So after each new rank that has been added the application will automatically update the attribute for the user.
- Add number_of_orders field to the Trip. Since in the UI we are going to show the number of orders takes for a trip again by doing so we will skip the aggregation on each request.
- Add deliverer attribute to Order. Since in the UI we are going to show the deliverer information for the Orders such way we will skip additional join with the Order->Trip->User table.

* Say a user has added a Trip to the system, then, he/she is going to browse through the orders, use some built-in filters according to data and itinerary to find an order to be delivered. Same scenario for the new Trip to be added.

In the next page you can see the our Preferred Relational Design.

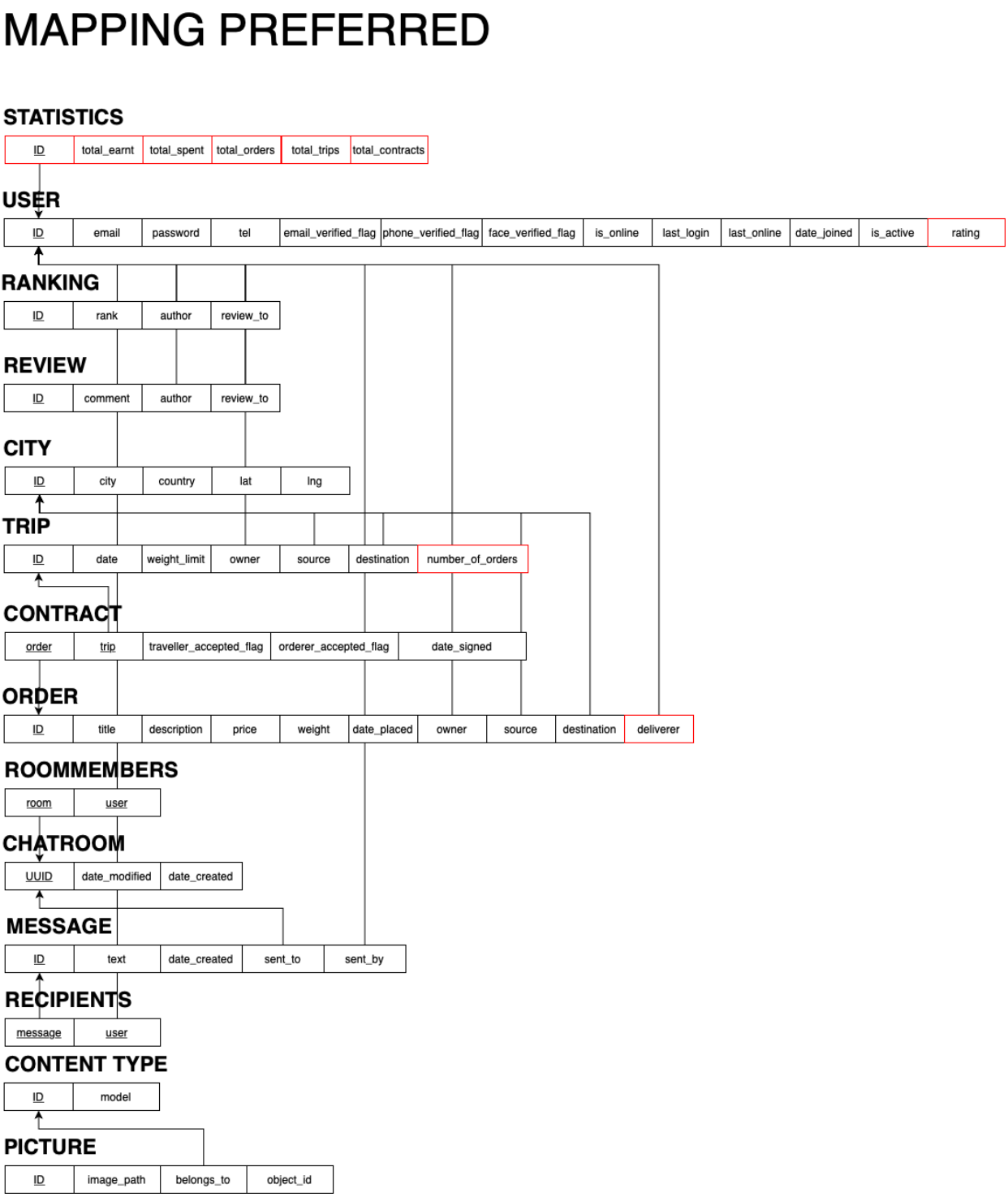

Figure 4https://drive.google.com/file/d/1mPJgPrvAwxgehIpCxDAgf3HiCO-ULtdj/view?usp=sharing

More About Content Type and Picture

Here is a sample file upload with the usage of type and picture.

User

11	q.rustam@code.edu.az	Rustam	Quliyev	+905338418625

ContentType

1	“User”

Picture

1	“static/image.jpg”	1	11

Hence This Picture<1> will be belonging to User with id of 11.

SQL Queries

The database of our choice is going to be PostgreSQL DBMS. We have chosen it for its great compatibility with Python, which is our language of choice for the Backend server. We are using Django framework for our Backend server, which has ORM(Object Relation Mapper). This feature makes it really easy to create, query and update the database, and makes the application highly scalable. However, for this part of the assignment we have dived deeper into raw queries of the Django framework and analyzed the performance of it. In the sections below we are going to display in essence what the framework does under the hood. And in the next chapter we are going to do some evaluation of such ORM technology.

Database Creation

Here are the queries for creating the database. The consists of 2 steps. First we are defining the tables with correct data types, then we are defining all the key constraints and indices.
NOTE: BriddgyCities class has auxiliary properties since it has been replicated from a dataset of cities.

```
BEGIN;
--
-- Create model BriddgyCities
--
CREATE TABLE "BriddgyApi_briddgycities" ("id" serial NOT NULL PRIMARY KEY, "city"
varchar(100) NOT NULL, "city_ascii" varchar(100) NOT NULL, "lat" double precision
NULL, "lng" double precision NULL, "country" varchar(100) NULL, "iso2" varchar(100)
NULL, "iso3" varchar(100) NULL, "admin_name" varchar(100) NULL, "capital" varchar(100)
NULL, "population" integer NULL, "id_ofcity" integer NULL);
--
-- Create model Room
--
CREATE TABLE "BriddgyApi_room" ("date_created" timestamp with time zone NOT NULL,
"date_modified" timestamp with time zone NOT NULL, "id" uuid NOT NULL PRIMARY KEY);
```

```
--
-- Create model User
--
CREATE TABLE "BriddgyApi_user" ("id" serial NOT NULL PRIMARY KEY, "password"
varchar(128) NOT NULL, "last_login" timestamp with time zone NULL, "is_superuser"
boolean NOT NULL, "is_number_verified" varchar(255) NOT NULL, "is_email_verified"
boolean NOT NULL, "is_photo_verified" boolean NOT NULL, "first_name" varchar(30) NOT
NULL, "last_name" varchar(150) NOT NULL, "email" varchar(254) NOT NULL UNIQUE,
"rating" double precision NULL, "is_staff" boolean NOT NULL, "is_active" boolean NOT
NULL, "date_joined" timestamp with time zone NOT NULL, "online" boolean NOT NULL,
"last_online" timestamp with time zone NOT NULL, "deviceToken" varchar(600) NOT NULL);
--
-- Create model Trip
--
CREATE TABLE "BriddgyApi_trip" ("id" serial NOT NULL PRIMARY KEY, "date" date NOT
NULL, "weight_limit" double precision NOT NULL, "number_of_contracts" integer NOT NULL
CHECK ("number_of_contracts" >= 0), "destination_id" integer NOT NULL, "owner_id"
integer NULL, "source_id" integer NOT NULL);
--
-- Create model Statistics
--
CREATE TABLE "BriddgyApi_statistics" ("id" serial NOT NULL PRIMARY KEY, "totaltrips"
integer NOT NULL CHECK ("totaltrips" >= 0), "totalcontracts" integer NOT NULL CHECK
("totalcontracts" >= 0), "totalearnings" integer NOT NULL CHECK ("totalearnings" >=
0), "totalorders" integer NOT NULL CHECK ("totalorders" >= 0), "totalspent" integer
NOT NULL CHECK ("totalspent" >= 0), "owner_id" integer NOT NULL UNIQUE);
--
-- Create model RoomMembers
--
CREATE TABLE "BriddgyApi_roommembers" ("id" serial NOT NULL PRIMARY KEY,
"unread_count" integer NOT NULL CHECK ("unread_count" >= 0), "online" boolean NOT
NULL, "room_id" uuid NOT NULL, "user_id" integer NOT NULL);
--
-- Create model Review
--
CREATE TABLE "BriddgyApi_review" ("id" serial NOT NULL PRIMARY KEY, "comment"
varchar(500) NULL, "author_id" integer NULL, "reviewTo_id" integer NULL);
--
-- Create model Ranking
--
CREATE TABLE "BriddgyApi_ranking" ("id" serial NOT NULL PRIMARY KEY, "rating" integer
NULL, "author_id" integer NULL, "reviewTo_id" integer NULL);
--
-- Create model Pictures
--
CREATE TABLE "BriddgyApi_pictures" ("id" serial NOT NULL PRIMARY KEY, "image"
varchar(100) NOT NULL, "object_id" integer NOT NULL CHECK ("object_id" >= 0),
"content_type_id" integer NOT NULL);
```

```
--
-- Create model Order
--
CREATE TABLE "BriddgyApi_order" ("id" serial NOT NULL PRIMARY KEY, "title"
varchar(255) NULL, "date" date NOT NULL, "weight" double precision NOT NULL, "address"
varchar(255) NOT NULL, "dimensions" double precision NOT NULL, "description"
varchar(255) NOT NULL, "price" double precision NOT NULL, "deliverer_id" integer NULL,
"destination_id" integer NULL, "owner_id" integer NULL, "source_id" integer NULL,
"trip_id" integer NULL);
--
-- Create model Notification
--
CREATE TABLE "BriddgyApi_notification" ("id" serial NOT NULL PRIMARY KEY, "content"
varchar(255) NOT NULL, "date" date NOT NULL, "read" boolean NOT NULL, "owner_id"
integer NOT NULL UNIQUE);
--
-- Create model Message
--
CREATE TABLE "BriddgyApi_message" ("id" serial NOT NULL PRIMARY KEY, "date_created"
timestamp with time zone NOT NULL, "date_modified" timestamp with time zone NOT NULL,
"text" text NOT NULL, "room_id" uuid NOT NULL, "sender_id" integer NOT NULL);
--
-- Create model Message Recipients
--
CREATE TABLE "BriddgyApi_message_recipients" ("id" serial NOT NULL PRIMARY KEY,
"message_id" integer NOT NULL, "user_id" integer NOT NULL);
--
-- Create model EmailNotification
--
CREATE TABLE "BriddgyApi_emailnotification" ("id" serial NOT NULL PRIMARY KEY,
"content" varchar(255) NOT NULL, "date" date NOT NULL, "from_room_id" uuid NOT NULL
UNIQUE, "owner_id" integer NOT NULL UNIQUE);
--
-- Create model Contracts
--
CREATE TABLE "BriddgyApi_contracts" ("id" serial NOT NULL PRIMARY KEY, "owner1"
integer NULL, "owner2" integer NULL, "IsTravelerAccepted" boolean NOT NULL,
"IsOrdererAccepted" boolean NOT NULL, "dateSigned" date NOT NULL, "order_id" integer
NULL, "trip_id" integer NULL);
--
-- Create model Content Types
--

CREATE TABLE "django_content_type" ("id" serial NOT NULL PRIMARY KEY, "name"
varchar(100) NOT NULL, "app_label" varchar(100) NOT NULL, "model" varchar(100) NOT
NULL);
```

```
--
-- Create model Email Confirmation
--

CREATE TABLE "simple_email_confirmation_emailaddress" ("id" serial NOT NULL PRIMARY
KEY, "email" varchar(255) NOT NULL, "key" varchar(40) NOT NULL UNIQUE, "set_at"
timestamp with time zone NOT NULL, "confirmed_at" timestamp with time zone NULL,
"user_id" integer NOT NULL);

--
-- Next we are going to define the Foreign keys, constraints and Indices!
--

ALTER TABLE "django_content_type" ADD CONSTRAINT
"django_content_type_app_label_model_76bd3d3b_uniq" UNIQUE ("app_label", "model");

ALTER TABLE "simple_email_confirmation_emailaddress" ADD CONSTRAINT
"simple_email_confirmatio_user_id_email_28a01714_uniq" UNIQUE ("user_id", "email");

ALTER TABLE "simple_email_confirmation_emailaddress" ADD CONSTRAINT
"simple_email_confirm_user_id_b0e04c62_fk_BriddgyAp" FOREIGN KEY ("user_id")
REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

CREATE INDEX "simple_email_confirmation_emailaddress_key_22d3e56e_like" ON
"simple_email_confirmation_emailaddress" ("key" varchar_pattern_ops);

CREATE INDEX "simple_email_confirmation_emailaddress_user_id_b0e04c62" ON
"simple_email_confirmation_emailaddress" ("user_id");

CREATE INDEX "BriddgyApi_user_email_0bf2a97c_like" ON "BriddgyApi_user" ("email"
varchar_pattern_ops);

ALTER TABLE "BriddgyApi_trip" ADD CONSTRAINT
"BriddgyApi_trip_destination_id_d2cc61b0_fk_BriddgyAp" FOREIGN KEY ("destination_id")
REFERENCES "BriddgyApi_briddgycities" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_trip" ADD CONSTRAINT
"BriddgyApi_trip_owner_id_ac6a1d10_fk_BriddgyApi_user_id" FOREIGN KEY ("owner_id")
REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_trip" ADD CONSTRAINT
"BriddgyApi_trip_source_id_b4a1210e_fk_BriddgyAp" FOREIGN KEY ("source_id") REFERENCES
"BriddgyApi_briddgycities" ("id") DEFERRABLE INITIALLY DEFERRED;
```

```
CREATE INDEX "BriddgyApi_trip_destination_id_d2cc61b0" ON "BriddgyApi_trip"
("destination_id");

CREATE INDEX "BriddgyApi_trip_owner_id_ac6a1d10" ON "BriddgyApi_trip" ("owner_id");

CREATE INDEX "BriddgyApi_trip_source_id_b4a1210e" ON "BriddgyApi_trip" ("source_id");

ALTER TABLE "BriddgyApi_statistics" ADD CONSTRAINT
"BriddgyApi_statistics_owner_id_86b78c08_fk_BriddgyApi_user_id" FOREIGN KEY
("owner_id") REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_roommembers" ADD CONSTRAINT
"BriddgyApi_roommembers_room_id_1feda82c_fk_BriddgyApi_room_id" FOREIGN KEY
("room_id") REFERENCES "BriddgyApi_room" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_roommembers" ADD CONSTRAINT
"BriddgyApi_roommembers_user_id_1612ee60_fk_BriddgyApi_user_id" FOREIGN KEY
("user_id") REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

CREATE INDEX "BriddgyApi_roommembers_room_id_1feda82c" ON "BriddgyApi_roommembers"
("room_id");

CREATE INDEX "BriddgyApi_roommembers_user_id_1612ee60" ON "BriddgyApi_roommembers"
("user_id");

ALTER TABLE "BriddgyApi_review" ADD CONSTRAINT
"BriddgyApi_review_author_id_761b77e2_fk_BriddgyApi_user_id" FOREIGN KEY ("author_id")
REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_review" ADD CONSTRAINT
"BriddgyApi_review_reviewTo_id_012c4df9_fk_BriddgyApi_user_id" FOREIGN KEY
("reviewTo_id") REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

CREATE INDEX "BriddgyApi_review_author_id_761b77e2" ON "BriddgyApi_review"
("author_id");

CREATE INDEX "BriddgyApi_review_reviewTo_id_012c4df9" ON "BriddgyApi_review"
("reviewTo_id");

ALTER TABLE "BriddgyApi_ranking" ADD CONSTRAINT
"BriddgyApi_ranking_author_id_a74e5a9a_fk_BriddgyApi_user_id" FOREIGN KEY
("author_id") REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_ranking" ADD CONSTRAINT
"BriddgyApi_ranking_reviewTo_id_bdb303c5_fk_BriddgyApi_user_id" FOREIGN KEY
("reviewTo_id") REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;
```

```
CREATE INDEX "BriddgyApi_ranking_author_id_a74e5a9a" ON "BriddgyApi_ranking"
("author_id");

CREATE INDEX "BriddgyApi_ranking_reviewTo_id_bdb303c5" ON "BriddgyApi_ranking"
("reviewTo_id");

ALTER TABLE "BriddgyApi_pictures" ADD CONSTRAINT
"BriddgyApi_pictures_content_type_id_4f22be9c_fk_django_co" FOREIGN KEY
("content_type_id") REFERENCES "django_content_type" ("id") DEFERRABLE INITIALLY
DEFERRED;

CREATE INDEX "BriddgyApi_pictures_content_type_id_4f22be9c" ON "BriddgyApi_pictures"
("content_type_id");

ALTER TABLE "BriddgyApi_order" ADD CONSTRAINT
"BriddgyApi_order_deliverer_id_1aa6b26c_fk_BriddgyApi_trip_id" FOREIGN KEY
("deliverer_id") REFERENCES "BriddgyApi_trip" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_order" ADD CONSTRAINT
"BriddgyApi_order_destination_id_ae294bbb_fk_BriddgyAp" FOREIGN KEY ("destination_id")
REFERENCES "BriddgyApi_briddgycities" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_order" ADD CONSTRAINT
"BriddgyApi_order_owner_id_b9373b2c_fk_BriddgyApi_user_id" FOREIGN KEY ("owner_id")
REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_order" ADD CONSTRAINT
"BriddgyApi_order_source_id_d4312248_fk_BriddgyAp" FOREIGN KEY ("source_id")
REFERENCES "BriddgyApi_briddgycities" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_order" ADD CONSTRAINT
"BriddgyApi_order_trip_id_2909561f_fk_BriddgyApi_trip_id" FOREIGN KEY ("trip_id")
REFERENCES "BriddgyApi_trip" ("id") DEFERRABLE INITIALLY DEFERRED;

CREATE INDEX "BriddgyApi_order_deliverer_id_1aa6b26c" ON "BriddgyApi_order"
("deliverer_id");

CREATE INDEX "BriddgyApi_order_destination_id_ae294bbb" ON "BriddgyApi_order"
("destination_id");

CREATE INDEX "BriddgyApi_order_owner_id_b9373b2c" ON "BriddgyApi_order" ("owner_id");

CREATE INDEX "BriddgyApi_order_source_id_d4312248" ON "BriddgyApi_order"
("source_id");

CREATE INDEX "BriddgyApi_order_trip_id_2909561f" ON "BriddgyApi_order" ("trip_id");
```

```
ALTER TABLE "BriddgyApi_notification" ADD CONSTRAINT
"BriddgyApi_notification_owner_id_261759f5_fk_BriddgyApi_user_id" FOREIGN KEY
("owner_id") REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_message" ADD CONSTRAINT
"BriddgyApi_message_room_id_de9adf61_fk_BriddgyApi_room_id" FOREIGN KEY ("room_id")
REFERENCES "BriddgyApi_room" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_message" ADD CONSTRAINT
"BriddgyApi_message_sender_id_05f41d52_fk_BriddgyApi_user_id" FOREIGN KEY
("sender_id") REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

CREATE INDEX "BriddgyApi_message_room_id_de9adf61" ON "BriddgyApi_message"
("room_id");

CREATE INDEX "BriddgyApi_message_sender_id_05f41d52" ON "BriddgyApi_message"
("sender_id");

ALTER TABLE "BriddgyApi_message_recipients" ADD CONSTRAINT
"BriddgyApi_message_r_message_id_810799ee_fk_BriddgyAp" FOREIGN KEY ("message_id")
REFERENCES "BriddgyApi_message" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_message_recipients" ADD CONSTRAINT
"BriddgyApi_message_r_user_id_0155d27f_fk_BriddgyAp" FOREIGN KEY ("user_id")
REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_message_recipients" ADD CONSTRAINT
"BriddgyApi_message_recipients_message_id_user_id_206f73a1_uniq" UNIQUE ("message_id",
"user_id");

CREATE INDEX "BriddgyApi_message_recipients_message_id_810799ee" ON
"BriddgyApi_message_recipients" ("message_id");

CREATE INDEX "BriddgyApi_message_recipients_user_id_0155d27f" ON
"BriddgyApi_message_recipients" ("user_id");

ALTER TABLE "BriddgyApi_emailnotification" ADD CONSTRAINT
"BriddgyApi_emailnoti_from_room_id_184bcc5f_fk_BriddgyAp" FOREIGN KEY ("from_room_id")
REFERENCES "BriddgyApi_room" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_emailnotification" ADD CONSTRAINT
"BriddgyApi_emailnoti_owner_id_64e17f2c_fk_BriddgyAp" FOREIGN KEY ("owner_id")
REFERENCES "BriddgyApi_user" ("id") DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "BriddgyApi_contracts" ADD CONSTRAINT
"BriddgyApi_contracts_order_id_b5b7cd99_fk_BriddgyApi_order_id" FOREIGN KEY
("order_id") REFERENCES "BriddgyApi_order" ("id") DEFERRABLE INITIALLY DEFERRED;
```

```
ALTER TABLE "BriddgyApi_contracts" ADD CONSTRAINT
"BriddgyApi_contracts_trip_id_f6507401_fk_BriddgyApi_trip_id" FOREIGN KEY ("trip_id")
REFERENCES "BriddgyApi_trip" ("id") DEFERRABLE INITIALLY DEFERRED;

CREATE INDEX "BriddgyApi_contracts_order_id_b5b7cd99" ON "BriddgyApi_contracts"
("order_id");

CREATE INDEX "BriddgyApi_contracts_trip_id_f6507401" ON "BriddgyApi_contracts"
("trip_id");

COMMIT;
```

Database Modifying Queries

Below are 3 simple database modifying queries.
NOTE: Since we are using Token based Authorization, most of the time the transaction starts off getting the user data from token info.

1. User Login Transaction Pipeline

```
--
--Create a new User
--
INSERT INTO "BriddgyApi_user" ("password", "last_login", "is_superuser",
"is_number_verified", "is_email_verified", "is_photo_verified", "first_name",
"last_name", "email", "rating", "is_staff", "is_active", "date_joined", "online",
"last_online", "deviceToken")
VALUES
('pbkdf2_sha256$150000$vOg5Y46piCet$O6dtrp0a4yQNso2l7XUav4byO7ouZ6C+oJUf0Va8jXM=',
NULL, false, 'false', false, false, 'Rustam', 'Quliyev', 'q.rustam@code.edu.az', 0.0,
false, false, '2020-04-24T12:14:39.440031+00:00'::timestamptz, false, '2020-04-
24T12:14:39.440042+00:00'::timestamptz, 'None') RETURNING "BriddgyApi_user"."id"
--
--Create new User Statistics
--
INSERT INTO "BriddgyApi_statistics" ("totaltrips", "totalcontracts", "totalearnings",
"totalorders", "totalspent", "owner_id") VALUES (0, 0, 0, 0, 0, 1) RETURNING
"BriddgyApi_statistics"."id"
--
--Create a new User Auth Token
--
INSERT INTO "authtoken_token" ("key", "user_id", "created")
VALUES ('cf93bc4955dcc417f698d4f40d97f54c15e6dedf', 1, '2020-04-
24T12:14:39.851388+00:00'::timestamptz)
--
--Create a new Email Confirmation
--
```

```
INSERT INTO "simple_email_confirmation_emailaddress" ("user_id", "email", "key",
"set_at", "confirmed_at")
VALUES (1, 'q.rustam@code.edu.az', 'IUwxBCpsEYjP', '2020-04-
24T12:14:39.668351+00:00'::timestamptz, NULL) RETURNING
"simple_email_confirmation_emailaddress"."id"
```

2. Create a Trip Query

```
--
--Select the User with Token
--
SELECT "authtoken_token"."key",
       "authtoken_token"."user_id",
       "authtoken_token"."created",
       "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "authtoken_token"
 INNER JOIN "BriddgyApi_user"
    ON ("authtoken_token"."user_id" = "BriddgyApi_user"."id")
 WHERE "authtoken_token"."key" = '988618c63be20b8f96a0dd27b8c2cc6f5538c83f'
--
--Get first City
--
SELECT "BriddgyApi_briddgycities"."id",
       "BriddgyApi_briddgycities"."city",
       "BriddgyApi_briddgycities"."city_ascii",
       "BriddgyApi_briddgycities"."lat",
       "BriddgyApi_briddgycities"."lng",
       "BriddgyApi_briddgycities"."country",
       "BriddgyApi_briddgycities"."iso2",
       "BriddgyApi_briddgycities"."iso3",
       "BriddgyApi_briddgycities"."admin_name",
       "BriddgyApi_briddgycities"."capital",
```

```
       "BriddgyApi_briddgycities"."population",
       "BriddgyApi_briddgycities"."id_ofcity"
  FROM "BriddgyApi_briddgycities"
 WHERE "BriddgyApi_briddgycities"."id" = 1
--
--Get second City
--
SELECT "BriddgyApi_briddgycities"."id",
       "BriddgyApi_briddgycities"."city",
       "BriddgyApi_briddgycities"."city_ascii",
       "BriddgyApi_briddgycities"."lat",
       "BriddgyApi_briddgycities"."lng",
       "BriddgyApi_briddgycities"."country",
       "BriddgyApi_briddgycities"."iso2",
       "BriddgyApi_briddgycities"."iso3",
       "BriddgyApi_briddgycities"."admin_name",
       "BriddgyApi_briddgycities"."capital",
       "BriddgyApi_briddgycities"."population",
       "BriddgyApi_briddgycities"."id_ofcity"
  FROM "BriddgyApi_briddgycities"
 WHERE "BriddgyApi_briddgycities"."id" = 2
--
--Insert Trip
--
INSERT INTO "BriddgyApi_trip" ("owner_id", "source_id", "destination_id", "date",
"weight_limit", "number_of_contracts")
VALUES (2, 1, 2, '2021-01-10'::date, 11.0, 0) RETURNING "BriddgyApi_trip"."id"
--
--Select Statistics
--
SELECT "BriddgyApi_statistics"."id",
       "BriddgyApi_statistics"."totaltrips",
       "BriddgyApi_statistics"."totalcontracts",
       "BriddgyApi_statistics"."totalearnings",
       "BriddgyApi_statistics"."totalorders",
       "BriddgyApi_statistics"."totalspent",
       "BriddgyApi_statistics"."owner_id"
  FROM "BriddgyApi_statistics"
 WHERE "BriddgyApi_statistics"."owner_id" = 2

--
--Update Statistics
--
 UPDATE "BriddgyApi_statistics"
   SET "totaltrips" = 1,
       "totalcontracts" = 0,
       "totalearnings" = 0,
       "totalorders" = 2,
```

```
       "totalspent" = 0,
       "owner_id" = 2
 WHERE "BriddgyApi_statistics"."id" = 2
```

3. Create an Order Query

```
--
--Select the User with Token
--
SELECT "authtoken_token"."key",
       "authtoken_token"."user_id",
       "authtoken_token"."created",
       "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "authtoken_token"
 INNER JOIN "BriddgyApi_user"
    ON ("authtoken_token"."user_id" = "BriddgyApi_user"."id")
 WHERE "authtoken_token"."key" = '2210ca9105391b73c06bc21e1b40cab084918c0b'
--
--Select the Source City existcance
--
 SELECT "BriddgyApi_briddgycities"."id",
       "BriddgyApi_briddgycities"."city",
       "BriddgyApi_briddgycities"."city_ascii",
       "BriddgyApi_briddgycities"."lat",
       "BriddgyApi_briddgycities"."lng",
       "BriddgyApi_briddgycities"."country",
       "BriddgyApi_briddgycities"."iso2",
       "BriddgyApi_briddgycities"."iso3",
       "BriddgyApi_briddgycities"."admin_name",
       "BriddgyApi_briddgycities"."capital",
       "BriddgyApi_briddgycities"."population",
```

```
       "BriddgyApi_briddgycities"."id_ofcity"
  FROM "BriddgyApi_briddgycities"
 WHERE "BriddgyApi_briddgycities"."id" = 1
--
--Select the Destination city existance
--
 SELECT "BriddgyApi_briddgycities"."id",
       "BriddgyApi_briddgycities"."city",
       "BriddgyApi_briddgycities"."city_ascii",
       "BriddgyApi_briddgycities"."lat",
       "BriddgyApi_briddgycities"."lng",
       "BriddgyApi_briddgycities"."country",
       "BriddgyApi_briddgycities"."iso2",
       "BriddgyApi_briddgycities"."iso3",
       "BriddgyApi_briddgycities"."admin_name",
       "BriddgyApi_briddgycities"."capital",
       "BriddgyApi_briddgycities"."population",
       "BriddgyApi_briddgycities"."id_ofcity"
  FROM "BriddgyApi_briddgycities"
 WHERE "BriddgyApi_briddgycities"."id" = 2
--
--CREATE ORder
--
 INSERT INTO "BriddgyApi_order" ("title", "owner_id", "source_id", "destination_id",
"date", "weight", "address", "dimensions", "trip_id", "description", "price",
"deliverer_id")
VALUES ('I need Kimono for Judo', 4, 1, 2, '2020-04-24'::date, 5.0, 'METU NCC', 22.0,
NULL, 'Kimono for Judo from Baku', 100.0, NULL) RETURNING "BriddgyApi_order"."id"
--
--Get User Stats
--
SELECT "BriddgyApi_statistics"."id",
       "BriddgyApi_statistics"."totaltrips",
       "BriddgyApi_statistics"."totalcontracts",
       "BriddgyApi_statistics"."totalearnings",
       "BriddgyApi_statistics"."totalorders",
       "BriddgyApi_statistics"."totalspent",
       "BriddgyApi_statistics"."owner_id"
  FROM "BriddgyApi_statistics"
 WHERE "BriddgyApi_statistics"."owner_id" = 4
--
--Update User stats
--
UPDATE "BriddgyApi_statistics"
   SET "totaltrips" = 1,
       "totalcontracts" = 0,
       "totalearnings" = 0,
       "totalorders" = 1,
```

```
       "totalspent" = 0,
       "owner_id" = 4
 WHERE "BriddgyApi_statistics"."id" = 4
```

Database Modification Queries

1. Display User (Transaction Pipeline)

```
--
--Select the User with id 1
--
SELECT "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "BriddgyApi_user"
 WHERE "BriddgyApi_user"."id" = 1
--
--Select the Picture of Users
--
SELECT "BriddgyApi_pictures"."id",
       "BriddgyApi_pictures"."image",
       "BriddgyApi_pictures"."content_type_id",
       "BriddgyApi_pictures"."object_id"
  FROM "BriddgyApi_pictures"
 WHERE ("BriddgyApi_pictures"."content_type_id" = 9 AND
"BriddgyApi_pictures"."object_id" = 1)
```

2. Get orders which are not mine (Transaction Pipeline)

```
--
--Select the User with Token
--
SELECT "authtoken_token"."key",
       "authtoken_token"."user_id",
       "authtoken_token"."created",
```

```
       "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "authtoken_token"
 INNER JOIN "BriddgyApi_user"
    ON ("authtoken_token"."user_id" = "BriddgyApi_user"."id")
 WHERE "authtoken_token"."key" = '2210ca9105391b73c06bc21e1b40cab084918c0b'
--
--Select the Orders which don't belong to current user
--
SELECT "BriddgyApi_order"."id",
       "BriddgyApi_order"."title",
       "BriddgyApi_order"."owner_id",
       "BriddgyApi_order"."source_id",
       "BriddgyApi_order"."destination_id",
       "BriddgyApi_order"."date",
       "BriddgyApi_order"."weight",
       "BriddgyApi_order"."address",
       "BriddgyApi_order"."dimensions",
       "BriddgyApi_order"."trip_id",
       "BriddgyApi_order"."description",
       "BriddgyApi_order"."price",
       "BriddgyApi_order"."deliverer_id"
  FROM "BriddgyApi_order"
 WHERE NOT ("BriddgyApi_order"."owner_id" = 4 AND "BriddgyApi_order"."owner_id" IS NOT
NULL)
--
--Select the Orders Images
--
 SELECT "BriddgyApi_pictures"."id",
       "BriddgyApi_pictures"."image",
       "BriddgyApi_pictures"."content_type_id",
       "BriddgyApi_pictures"."object_id"
  FROM "BriddgyApi_pictures"
```

```
 WHERE ("BriddgyApi_pictures"."content_type_id" = 16 AND
"BriddgyApi_pictures"."object_id" = 4)
--
--Select the Source City
--
SELECT "BriddgyApi_briddgycities"."id",
       "BriddgyApi_briddgycities"."city",
       "BriddgyApi_briddgycities"."city_ascii",
       "BriddgyApi_briddgycities"."lat",
       "BriddgyApi_briddgycities"."lng",
       "BriddgyApi_briddgycities"."country",
       "BriddgyApi_briddgycities"."iso2",
       "BriddgyApi_briddgycities"."iso3",
       "BriddgyApi_briddgycities"."admin_name",
       "BriddgyApi_briddgycities"."capital",
       "BriddgyApi_briddgycities"."population",
       "BriddgyApi_briddgycities"."id_ofcity"
  FROM "BriddgyApi_briddgycities"
 WHERE "BriddgyApi_briddgycities"."id" = 1
--
--Select the Destination City
--
 SELECT "BriddgyApi_briddgycities"."id",
       "BriddgyApi_briddgycities"."city",
       "BriddgyApi_briddgycities"."city_ascii",
       "BriddgyApi_briddgycities"."lat",
       "BriddgyApi_briddgycities"."lng",
       "BriddgyApi_briddgycities"."country",
       "BriddgyApi_briddgycities"."iso2",
       "BriddgyApi_briddgycities"."iso3",
       "BriddgyApi_briddgycities"."admin_name",
       "BriddgyApi_briddgycities"."capital",
       "BriddgyApi_briddgycities"."population",
       "BriddgyApi_briddgycities"."id_ofcity"
  FROM "BriddgyApi_briddgycities"
 WHERE "BriddgyApi_briddgycities"."id" = 2
```

3. Get count of all unread notifications

```
--
--Select the User with Token
--
SELECT "authtoken_token"."key",
       "authtoken_token"."user_id",
       "authtoken_token"."created",
       "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
```

```
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "authtoken_token"
 INNER JOIN "BriddgyApi_user"
    ON ("authtoken_token"."user_id" = "BriddgyApi_user"."id")
 WHERE "authtoken_token"."key" = '2210ca9105391b73c06bc21e1b40cab084918c0b'
--
--Count Unread Notifications
--
 SELECT COUNT(*) AS "__count"
FROM "BriddgyApi_notification"
 WHERE ("BriddgyApi_notification"."owner_id" = 4 AND "BriddgyApi_notification"."read"
= false)
```

4. Get chat rooms List

```
--
--Select the User with Token
--
SELECT "authtoken_token"."key",
       "authtoken_token"."user_id",
       "authtoken_token"."created",
       "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
```

```
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "authtoken_token"
 INNER JOIN "BriddgyApi_user"
    ON ("authtoken_token"."user_id" = "BriddgyApi_user"."id")
 WHERE "authtoken_token"."key" = '2210ca9105391b73c06bc21e1b40cab084918c0b'
--
--Select the User with whom we want to chat
--
 SELECT "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "BriddgyApi_user"
 WHERE "BriddgyApi_user"."id" = 3
--
--GET ROOMs
--
SELECT "BriddgyApi_room"."date_created",
       "BriddgyApi_room"."date_modified",
       "BriddgyApi_room"."id"
  FROM "BriddgyApi_room"
 INNER JOIN "BriddgyApi_roommembers"
    ON ("BriddgyApi_room"."id" = "BriddgyApi_roommembers"."room_id")
 WHERE "BriddgyApi_roommembers"."user_id" = 4
 ORDER BY "BriddgyApi_room"."date_modified" DESC
 --
 --Get room members details
 --
 SELECT "BriddgyApi_roommembers"."id",
       "BriddgyApi_roommembers"."room_id",
       "BriddgyApi_roommembers"."user_id",
       "BriddgyApi_roommembers"."unread_count",
```

```
       "BriddgyApi_roommembers"."online"
  FROM "BriddgyApi_roommembers"
 WHERE "BriddgyApi_roommembers"."room_id" = '44d6e2fb-4ade-4556-8fa6-
823c1f301bd0'::uuid
 --
 --Get room members User details
 --
 SELECT "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "BriddgyApi_user"
 WHERE "BriddgyApi_user"."id" = 3
 --
 --Get room members profile images
 --
SELECT "BriddgyApi_pictures"."id",
       "BriddgyApi_pictures"."image",
       "BriddgyApi_pictures"."content_type_id",
       "BriddgyApi_pictures"."object_id"
  FROM "BriddgyApi_pictures"
 WHERE ("BriddgyApi_pictures"."content_type_id" = 9 AND
"BriddgyApi_pictures"."object_id" = 3)
```

5. Get statistics about another user

```
--
-- Validate Auth User
--
SELECT "authtoken_token"."key",
       "authtoken_token"."user_id",
       "authtoken_token"."created",
       "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
```

```
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "authtoken_token"
 INNER JOIN "BriddgyApi_user"
    ON ("authtoken_token"."user_id" = "BriddgyApi_user"."id")
 WHERE "authtoken_token"."key" = '2210ca9105391b73c06bc21e1b40cab084918c0b'
 --
 --Get Requested Users details
 --
 SELECT "BriddgyApi_user"."id",
       "BriddgyApi_user"."password",
       "BriddgyApi_user"."last_login",
       "BriddgyApi_user"."is_superuser",
       "BriddgyApi_user"."is_number_verified",
       "BriddgyApi_user"."is_email_verified",
       "BriddgyApi_user"."is_photo_verified",
       "BriddgyApi_user"."first_name",
       "BriddgyApi_user"."last_name",
       "BriddgyApi_user"."email",
       "BriddgyApi_user"."rating",
       "BriddgyApi_user"."is_staff",
       "BriddgyApi_user"."is_active",
       "BriddgyApi_user"."date_joined",
       "BriddgyApi_user"."online",
       "BriddgyApi_user"."last_online",
       "BriddgyApi_user"."deviceToken"
  FROM "BriddgyApi_user"
 WHERE "BriddgyApi_user"."id" = 4
 --
 --Get Requested Users Statistics
 --
SELECT "BriddgyApi_statistics"."id",
       "BriddgyApi_statistics"."totaltrips",
       "BriddgyApi_statistics"."totalcontracts",
       "BriddgyApi_statistics"."totalearnings",
       "BriddgyApi_statistics"."totalorders",
```

```
      "BriddgyApi_statistics"."totalspent",
      "BriddgyApi_statistics"."owner_id"
  FROM "BriddgyApi_statistics"
 WHERE "BriddgyApi_statistics"."owner_id" = 4
```

Summary & Performance Review

In this section we are going to share the results & our reviews as to how well Django's ORM is capable of translating to SQL language.

Database Creation

We can see Django manages to create the database quite well. It defined the data types according to the models defined, and for each unique field and foreign key it tends to create a database index which is great for performance. PostgreSQL supports several types of constraints checking conditions & timings, those are *DEFERRABLE, NOT DEFERRABLE* with *INITIALLY IMMEDIATE, INITIALLY DEFERRED.* Those hyperparameters are used to specify when the given constraint is going to be checked & is it possible to defer the checking process. Django on the other hand tends to give DEFERRABLE INITIALLY DEFERED, which means that the checking on the constraint is happening only at the end of transaction.

Database Querying

As to what comes to querying the database we have found out that it is really like a double edged sword. The simplicity provided by the ORM makes it really easy to mess something up, and generally even the best frameworks for serializing objects tend to have redundant querying sometimes. Let's elaborate a bit more on the issues that we have found within the scope of our project.

Token Authentication

The framework's go to Token Authentication Middleware library is getting all the details of the requesting user on each request. That is it is selecting all the fields even if the transaction does not require it. This puts certain fixed overhead to the requests after user logges in.

Source & Destination City

Let's move our attention to the query for creating a new Trip or Order. Specifically for the point where we query Cities table. The purpose of that query is check the existence of the given source and destination cities in that table. The ORM decides to execute 2 separate SELECT operations one for the destination and other for source city. Moreover, it projects the results over the entire attributes which is also unnecessary. All of this could have been done with less projections and 1 SQL query.

The Problem of Listing

Let's now draw our attention to the order listing transaction. In the example provided our database had only 1 order therefore the framework did everything fine, except, the thing we talked about in the previous paragraph, it separated the source and destination fetching request into 2 select queries whereas it could have done it using and **or** predicate. However, we then added 1 more order to the system(total of 2 orders) and tried one more. Interestingly the framework was simply sending 3 selects for **each** order in the list of 2. So it was iterating

through the orders and trying fetch its images, source city and dest city all in separate queries. If we were to do it using raw SQL we would be sending all fetches in one request with a simple or predicate.

Finally for testing purposes we have run our own query in order to check the difference in performance that has occurred. In this case the database has 3 orders in the system.

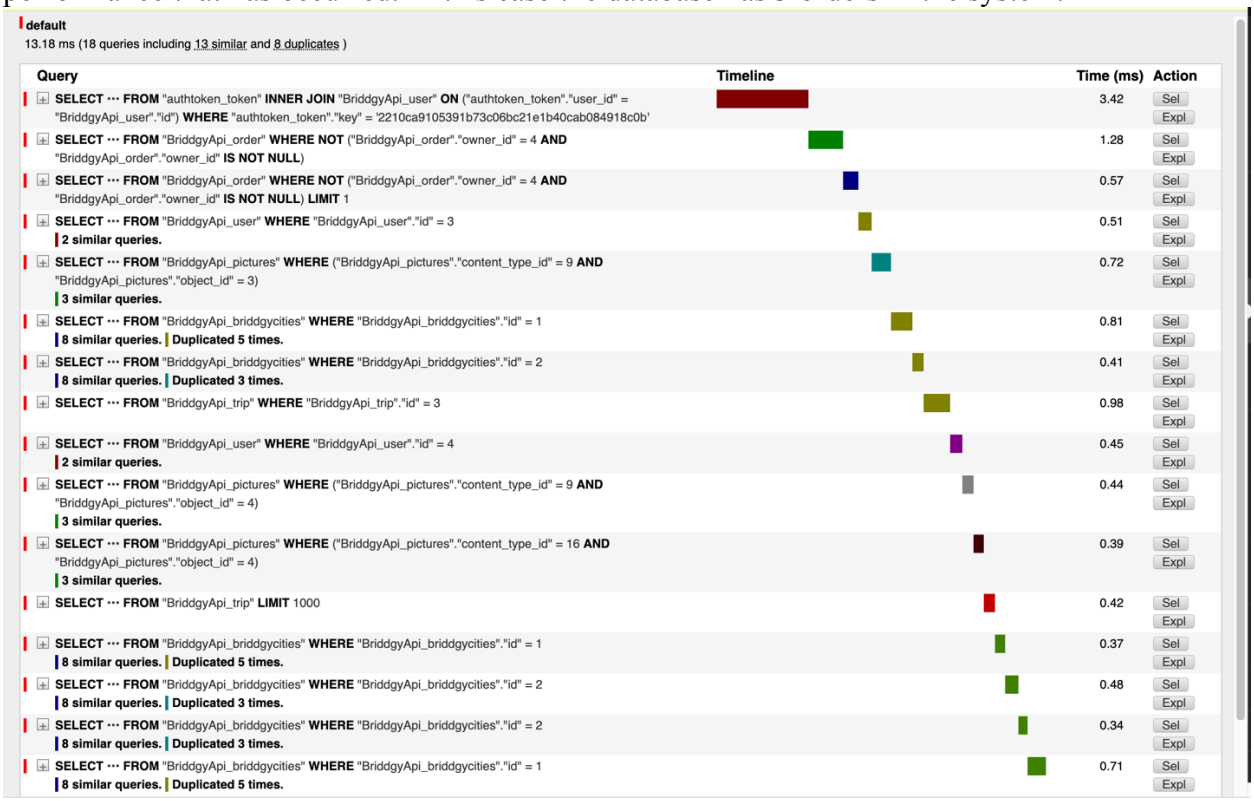

Figure 5 | Django's default querying system

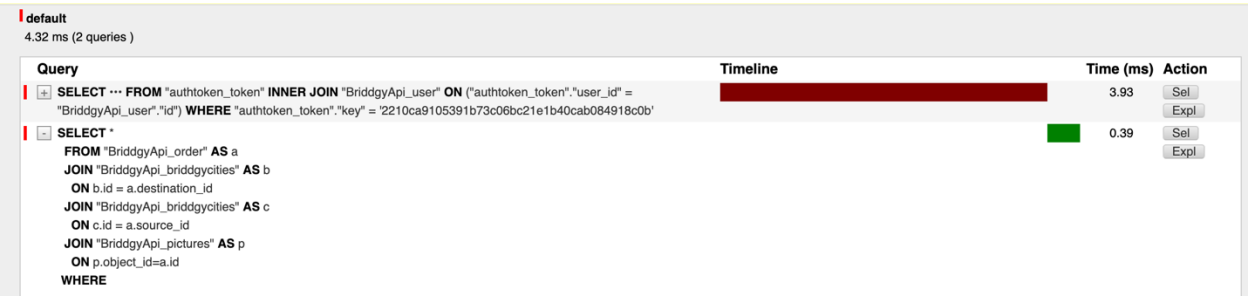

Figure 6 | Our query using 4 Joins

As you can see there is a drastic change in performance.

Summary

All in all such frameworks make it really easy to deploy the application and increase maintainability, however, it is really necessary to monitor the system and maybe use raw SQL queries sometimes.

Modifications & Graphical User Interface

Since last report there were no modifications in the Back-End side. As for the front-end we have created a Web Application on React, and deployed it to Heroku Cloud Provider. Here is the URL of the website: https://briddgy-web.herokuapp.com/en. Please use any browser other than Safari. There was a bug in Safari, with user login, that we could not yet figure out.

Screenshot Demo

Now we are going to include screenshots of the UI with their respective back-end functionalities, and steps that the Application performs.

1.Login

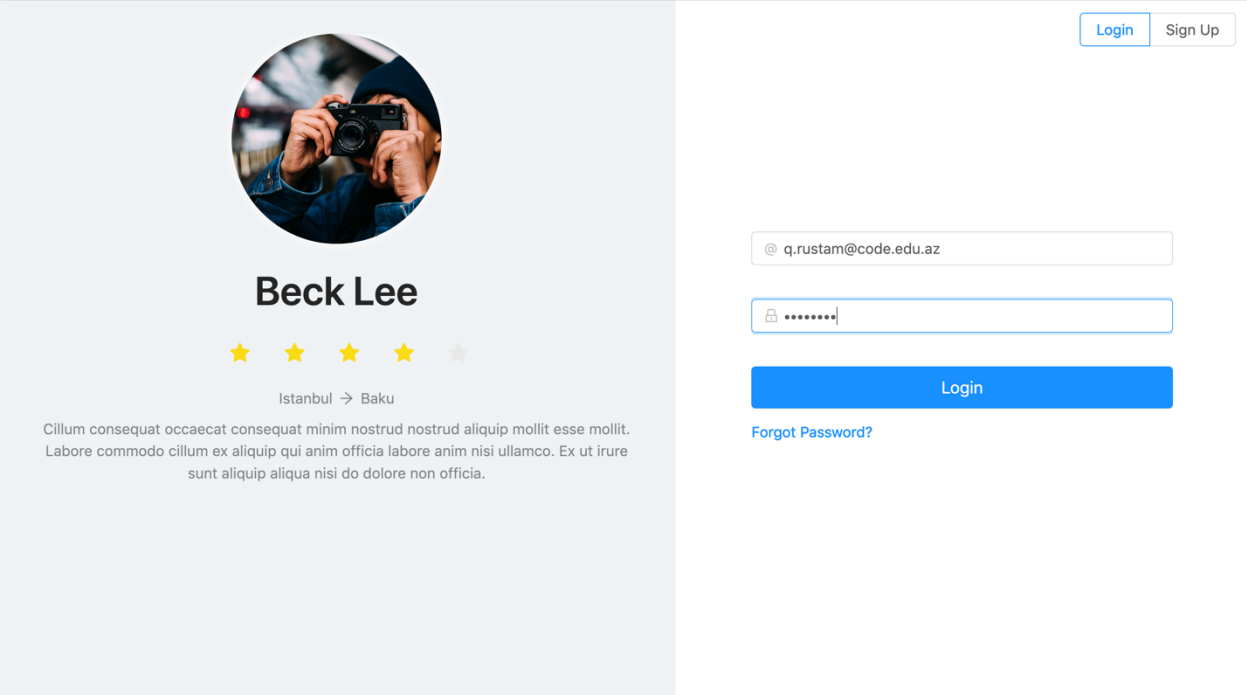

Figure 7 | Login Page

- Ask for User Token & populate in Cookie
- Get User's details from token
- Get Users chats

2. List all Orders which are not mine

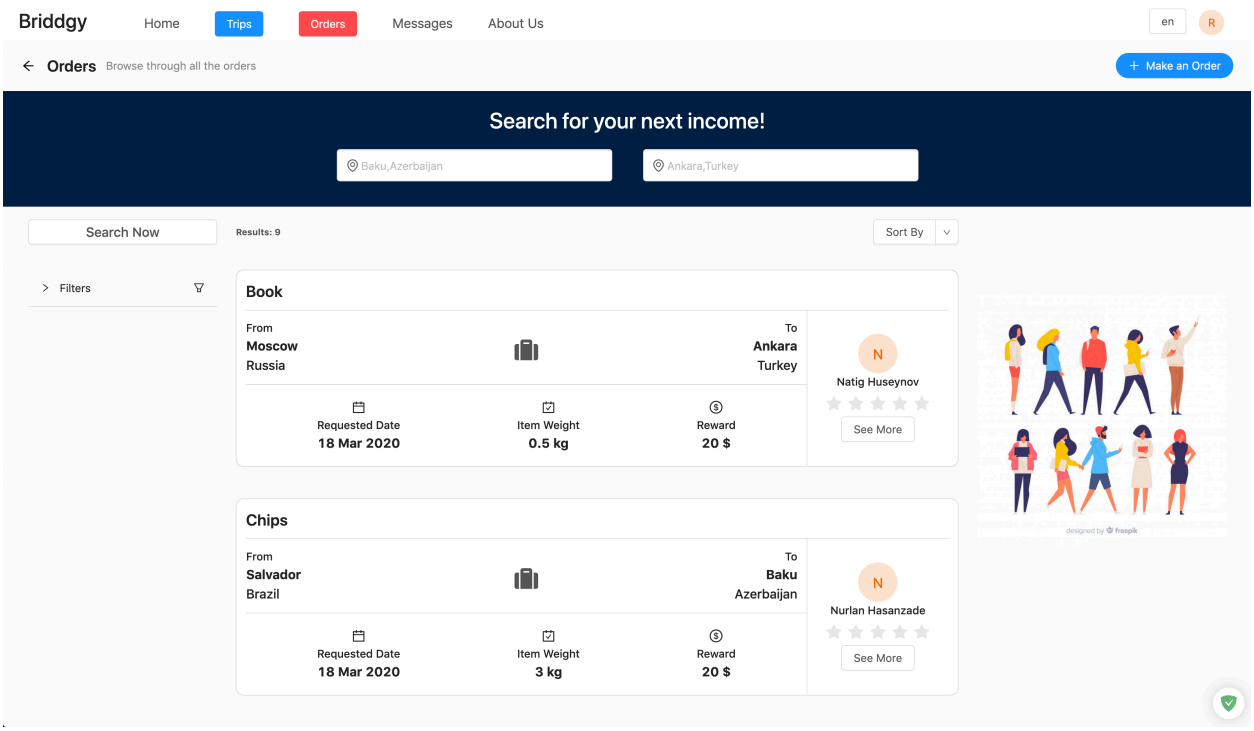

Figure 8 | List Orders View

All the filters on the left side are also working as well as the main Location Filter at the very top.

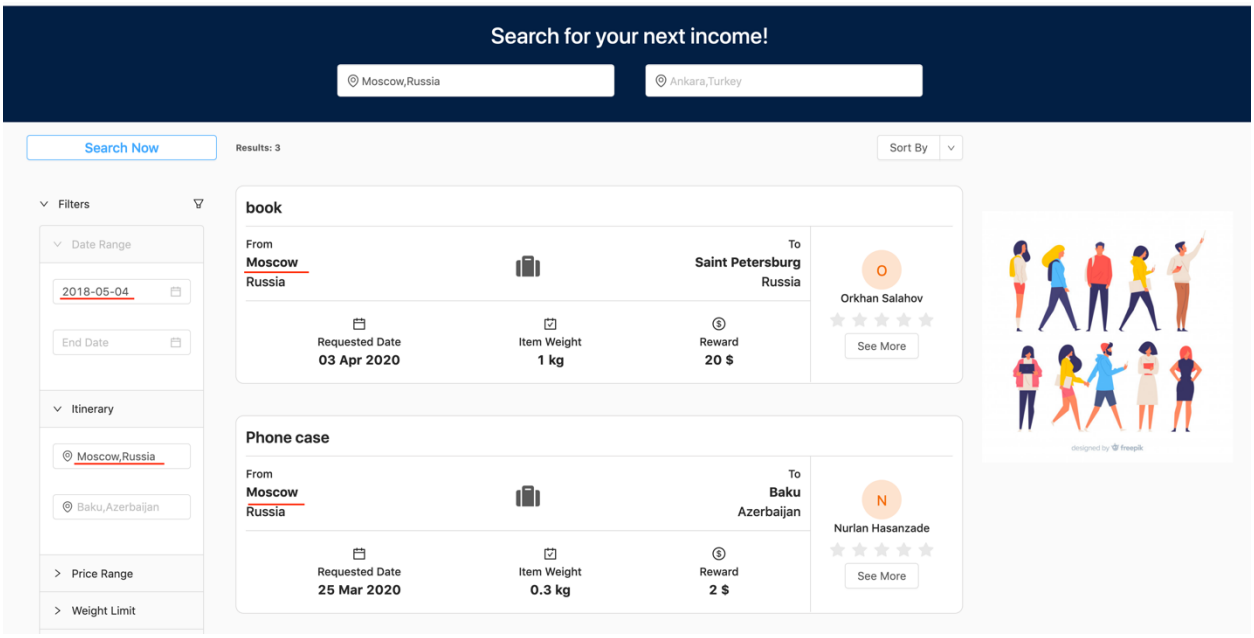

Figure 9 | List Orders Filtered View

3. List all Trips which are not mine

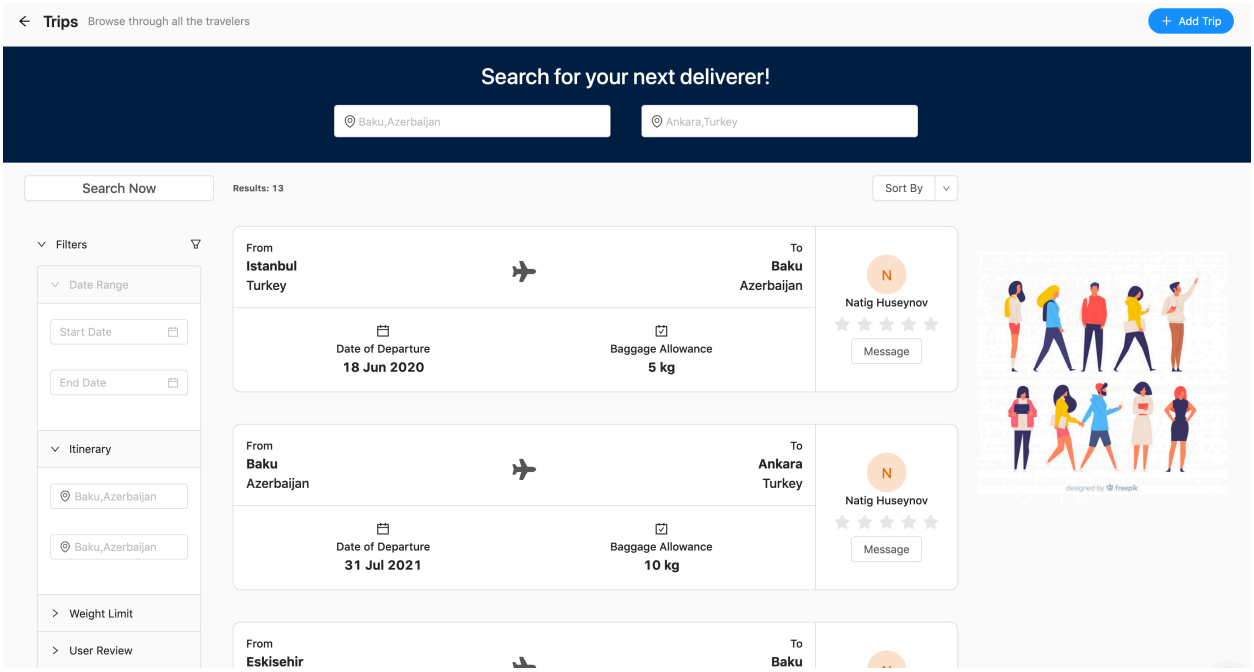

Figure 10 | List Trips View

4. My Profile

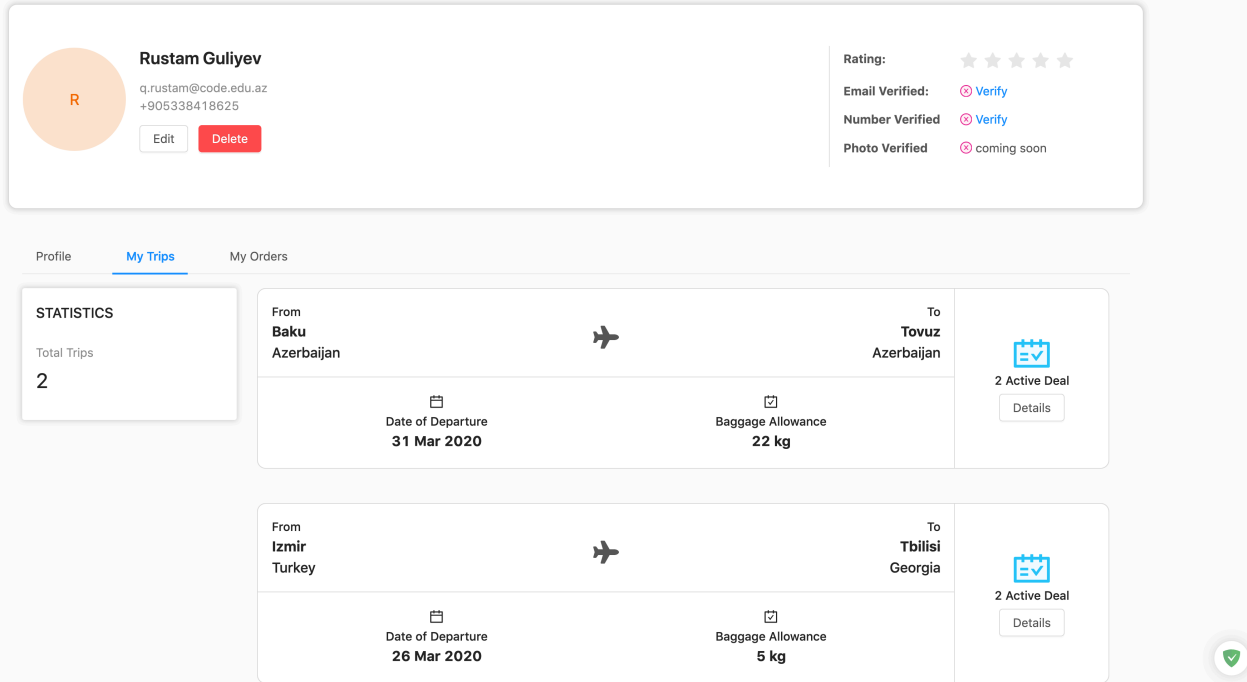

Figure 11 | My Profile View

- Fetch User Trips
- Fetch User Reviews
- Fetch User Orders
- Fetch User Statistics

5. Create a new Trip

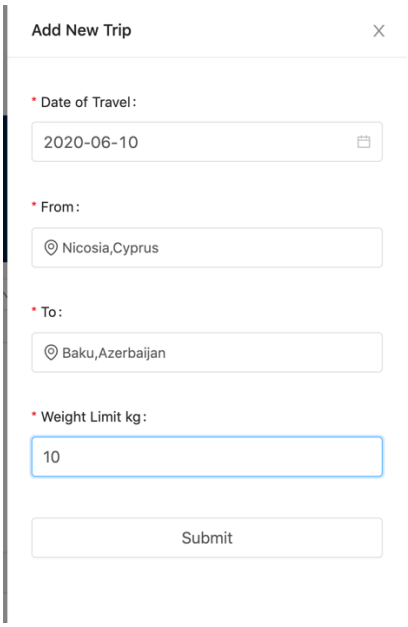

Figure 12 | Insert Trip Form

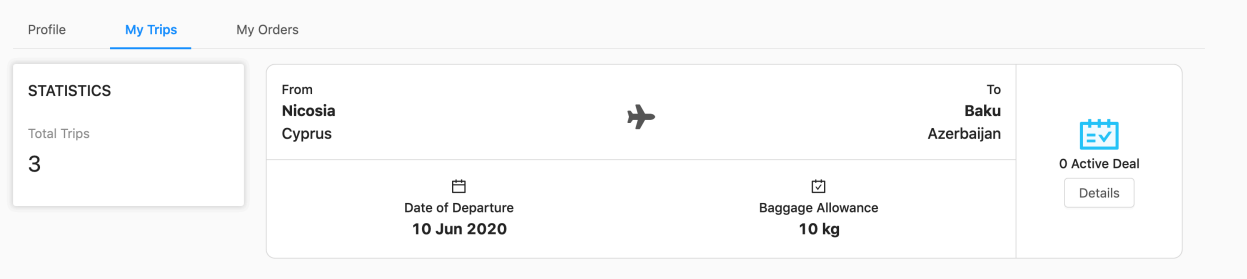

Figure 13 | Added Trip

6. Create a new Order

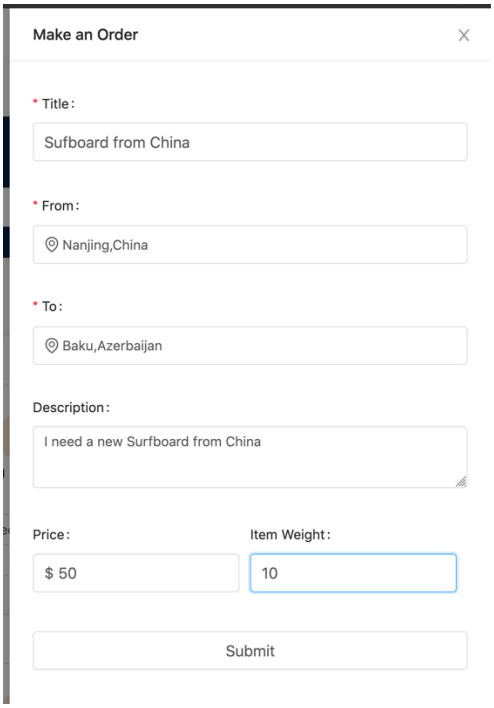

Figure 14 | Insert Order Form

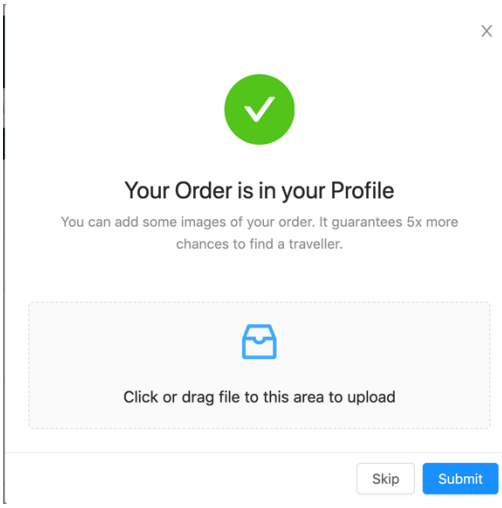

Figure 15 | Insert Order Image Upload

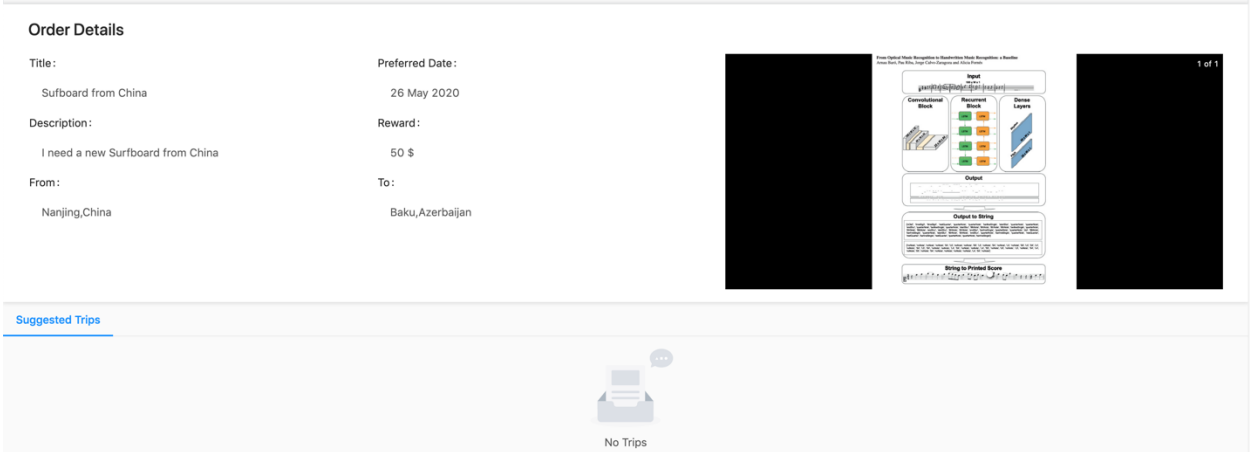

Figure 16 | Inserted Order

7. Messaging

One can start a new Chat Room with another users from:

- Entering to one's profile
- From List of trips
- From List of Orders

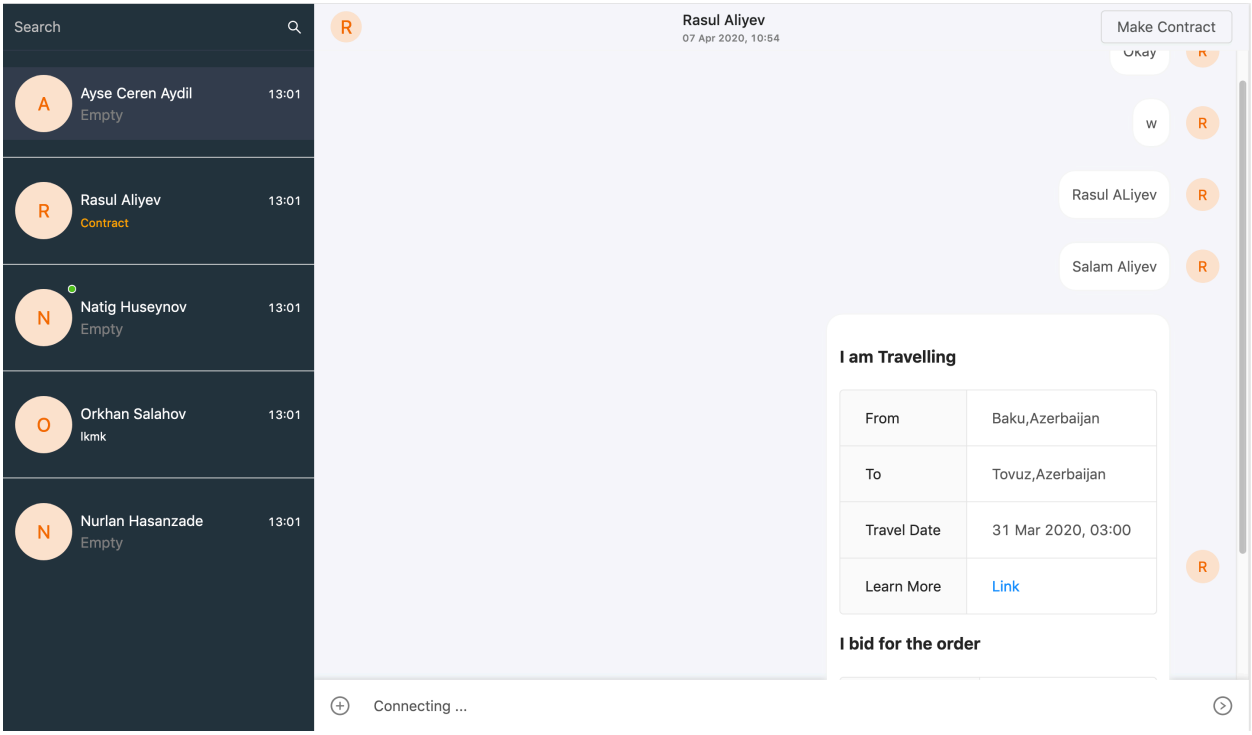

Figure 17 | Messaging UI

If you have a Trip or Order in the system & your interlocutor has a matching Order or Trip, you can create a new Contract with the User. We didn't include the Screenshots of Make Contract since there were multiple, small, pages. You can check them online through the provided link.

Also once the contract is created the system sends a message with prompt to Accept or Decline the offer. Once the interlocutor has accepted the contract is updated to both parties.

Mobile UI

Rasul and Orkhan have also developed Android & iOS Application for Briddgy. It is also fully functional that is why we decided to include the screenshots as well.

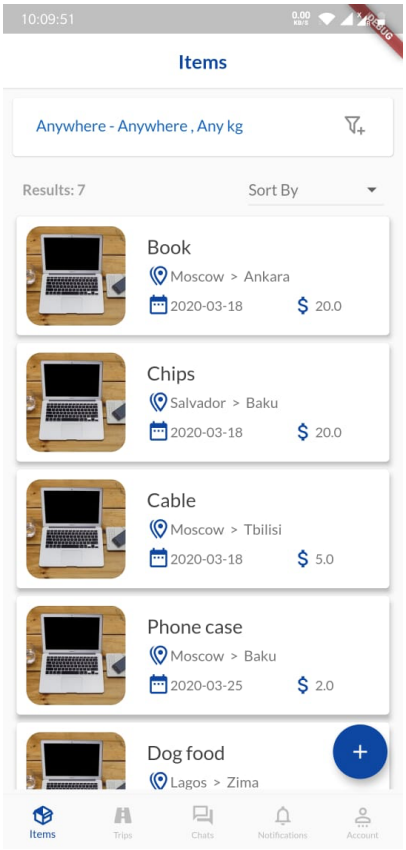

Figure 18 | List Orders

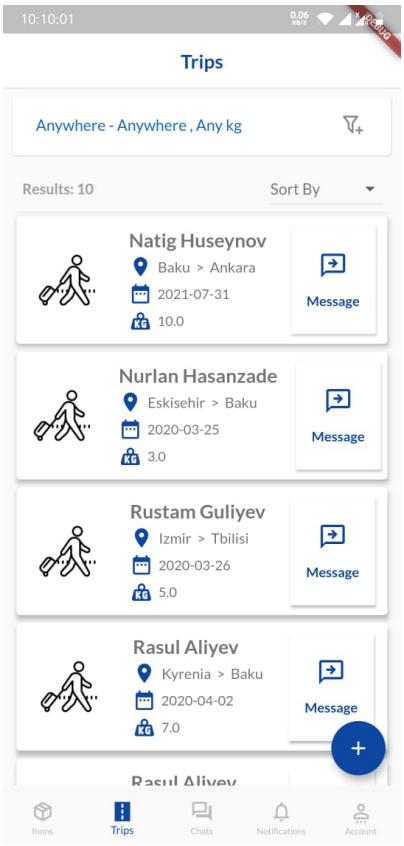

Figure 19 | List Trips

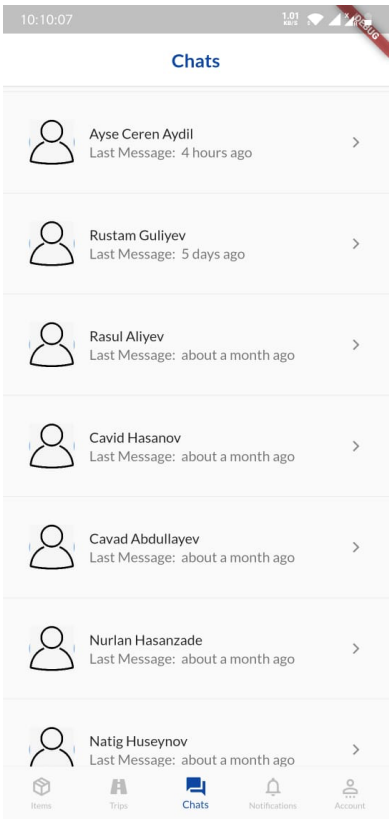

Figure 20 | Chat Page

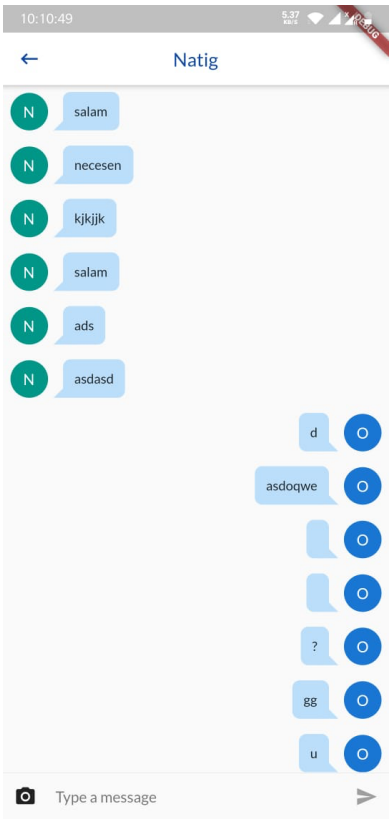

Figure 21 | Chat Page

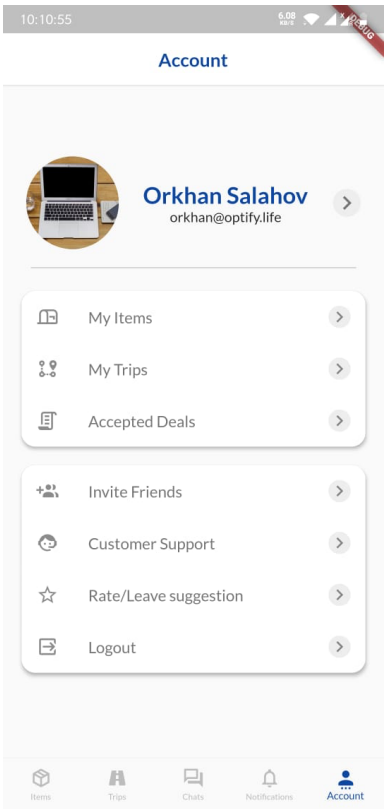

Figure 22 | Profile Page

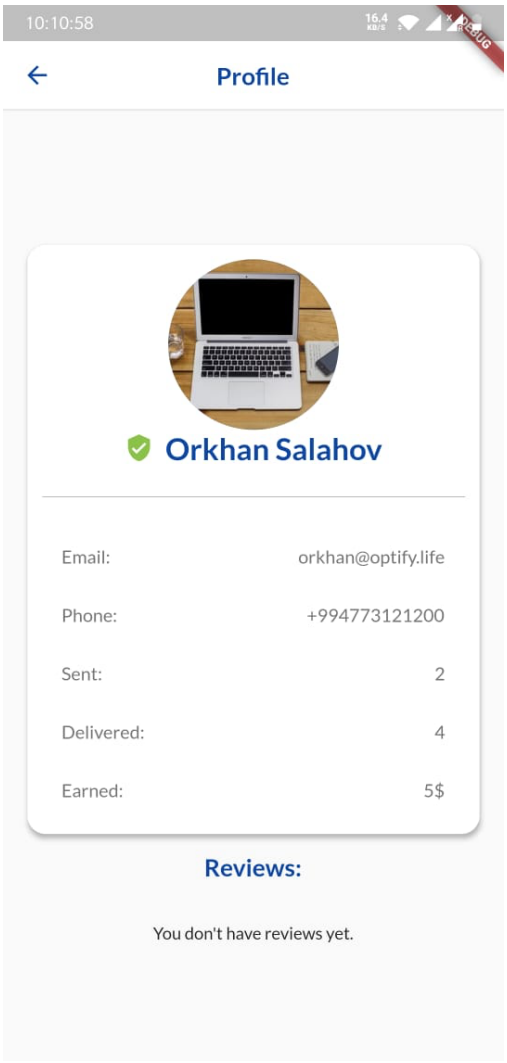

Figure 23 | Profile Info Page